

Contract-LIB

A Common Interchange Format for Software System Specification

Presentation at the Alpine Verification Meeting 2024, Freiburg im Breisgau

Draft: <https://www.sosy-lab.org/people/ernst/>

Code: <https://github.com/gernst/contract-lib>

To appear at SpecifyThis @ ISoLA 2024



Gidon Ernst



Wolfram Pfeifer
Mattias Ulbrich



Build-a-Deductive-Verifier Checklist

2/17

- 💡 have a new cool idea ✓
- ☕ target programming language ✓
- ★ foundational methodology ✓
- EA mathematical specification language ✓

(think: a tool like Dafny)

Build-a-Deductive-Verifier Checklist

3/17

- 💡 have a new cool idea ✓
- ☕ target programming language ✓
- ★ foundational methodology ✓
- ∀∃ mathematical specification language ✓
- 😊 have fun verifying programs

Build-a-Deductive-Verifier Checklist

4/17

- 💡 have a new cool idea ✓
- ☕ target programming language ✓
- ★ foundational methodology ✓
- EA mathematical specification language ✓
- 📁 support the standard library ... ⏳
- 🔌 share verified code / specs across tools ... !?
- 🔌 share proof artifacts across tools ... !?
- 🔌 connect proofs across languages ... !?



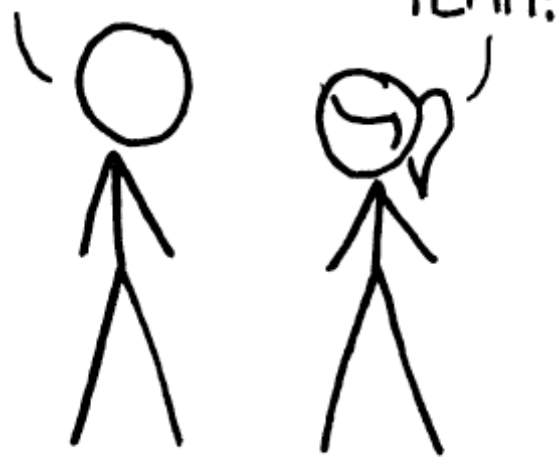
hindering progress
towards “real world”

HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)


SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

- On the wish-list since forever [e.g. Rushby 2005]
- Conflict
 - **innovation** yay cool research! [Separation Logic, ...]
 - **maturity** yay practical impact! [De Gouw+ 2015]
- Landscape of deductive verifiers
 -  tool lock-in, reinventing the wheel
 - \nexists standards are rather complex [JML, ACSL]

key challenge: find **simple** common ground

First Attempt: ArrayList in Dafny

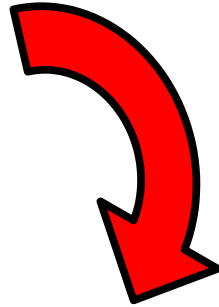
7/17

```
class ArrayList<T> {  
  var data: array<T>  
  var length: int  
  
  method add(last: T)  
    ensures length = old(length) + 1  
    ensures data[old(length)] = last  
    ensures forall i :: 0 ≤ i < old(length) ⇒  
      data[i] = old(data[i])  
}
```

First Attempt: ArrayList in Dafny

8/17

```
class ArrayList<T> {  
  var data: array<T>  
  var length: int
```



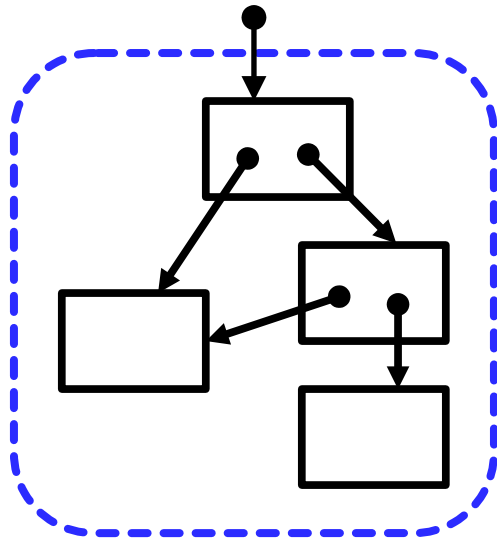
bad abstraction

- leak implementation details
- leak language semantics
- leak verification details

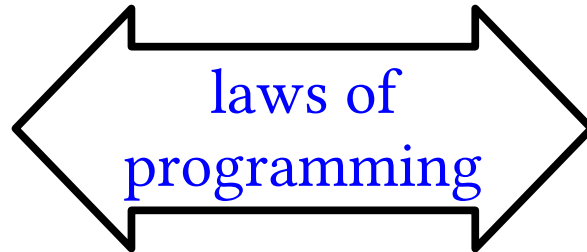
```
  method add(last: T)  
    ensures length = old(length) + 1  
    ensures data[old(length)] = last  
    ensures forall i :: 0 ≤ i < old(length) ⇒  
      data[i] = old(data[i])  
}
```


Software Verification (Principled Approach) 9/17

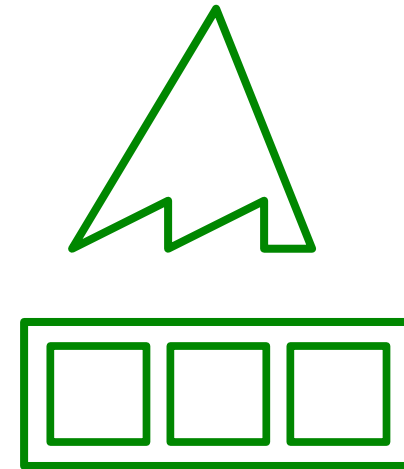
implementation



data structures
pointers etc, ...



specification



data abstractions
(sequences, sets,
maps, trees, ...)

Second Attempt: ArrayList in Dafny

10/17

```
class ArrayList<T> {  
  var data: array<T>  
  var length: int  
  ghost var content: seq<T>  
  
  method add(last: T)  
    ensures content = old(content) + [last]  
    requires valid()  
    ensures valid()  
  
  predicate valid() { ... }  
}
```

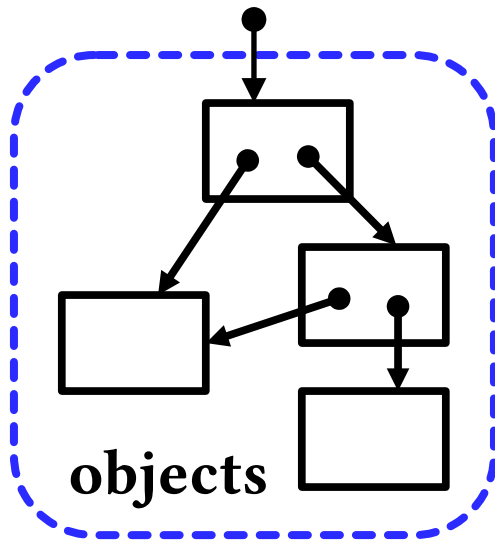
- **behavior: easy & intuitive**
- **implementation as well as abstraction mechanism is private to the class**

Key Observation

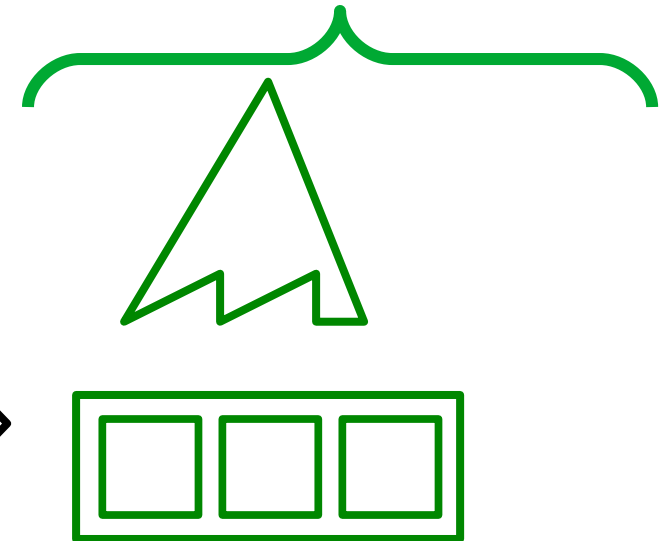
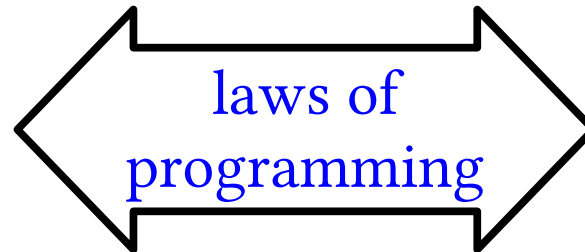
11/17

**complex & difficult
innovation happens here**

 **well-understood
stable across tools**



**data structures
pointers etc, ...**



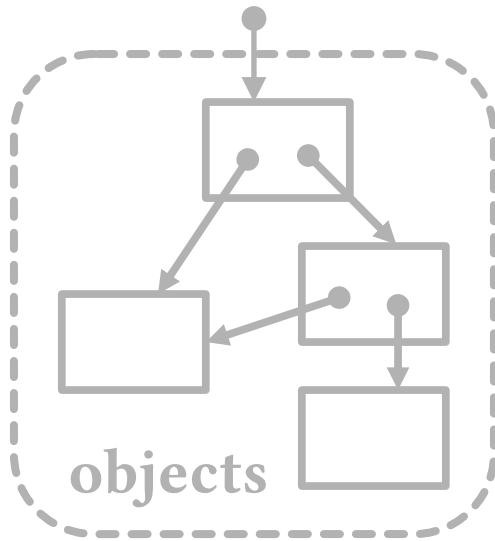
**data abstractions
(sequences, sets,
maps, trees, ...)**

Lesson from SMT-LIB etc...:

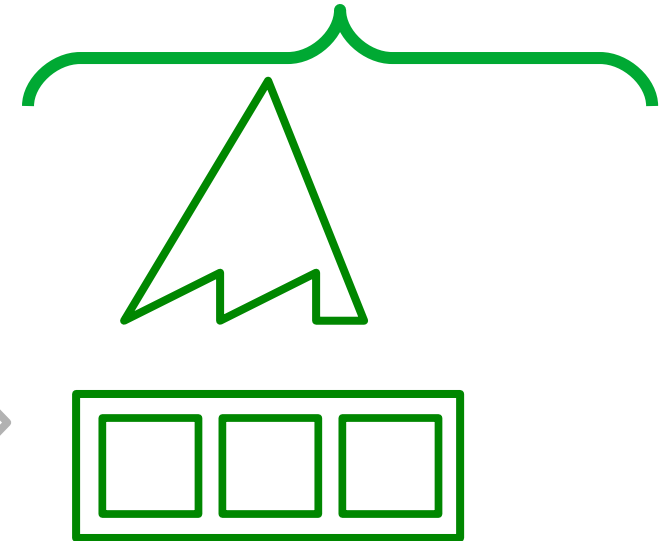
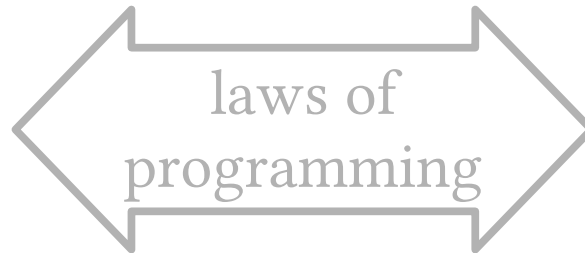
12/17

Have a clear & well-defined scope

🧠 **well-understood
stable across tools**



data structures
pointers etc, ...



**data abstractions
(sequences, sets,
maps, trees, ...)**

- **identify simple common ground**
 - behaviors of well-encapsulated stateful components / OOP
 - be independent of language, method, tool
 - but be universally compatible (... maybe not with Rust)

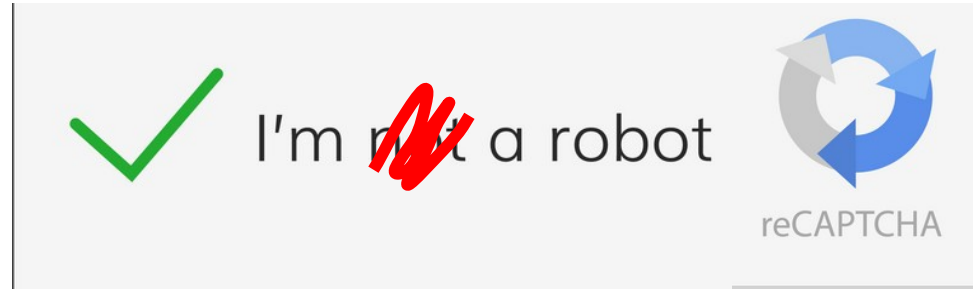
- **identify simple common ground**
 - behaviors of well-encapsulated stateful components / OOP
 - be independent of language, method, tool
 - but be universally compatible (... maybe not with Rust)
- **define easy-to-adopt technical realization: SMT-LIB with**
 - **old, par**, standardize extra theories (**Map, Seq, Set**)
 - **declare-abstractions** data model of abstract state
 - **define-contract** behavioral model for methods

ArrayList **Interface** in Contract-LIB

15/17

(declare-abstractions

```
((ArrayList 1))  
(par (T)  
  ((ArrayList  
    (ArrayList.content (Seq T))))))
```



(define-contract

```
ArrayList.add  
(par (T) ((this (inout ArrayList)) (last (in T)))  
  ((true  
    (= (ArrayList.content this)  
      (seq.++ (old (ArrayList.content this))  
              (seq.unit last))))))
```

Contract-LIB: define and formalize common ground of specifications across deductive verification tools

- Ongoing: Tool-chain (Java) + Adoption Guidelines
 - Outlook: integrate with tools & gather experience
 - share specs / verified components across tools
 - develop a common repository of case-studies and standard libraries
 - Paper: design discussion, examples, semantics, integration guideline, related work (MoXI, CHC, Boogie, Why3, ...)
- ♥ Acknowledgement: Thanks to all participants in the intense discussions at Dagstuhl and Lorentz seminars

Integration Workflow

17/17

